BCA611 Bilgisayar Grafiği

Ders 8 : Particle simulation

Zümra Kavafoğlu

Physics Simulation

Physics Simulation

- All motion in interactive physics-based animation is the result of on-line physics simulation.

- A physics simulator iteratively updates the state of a virtual environment, based on its current state, and external forces and torques

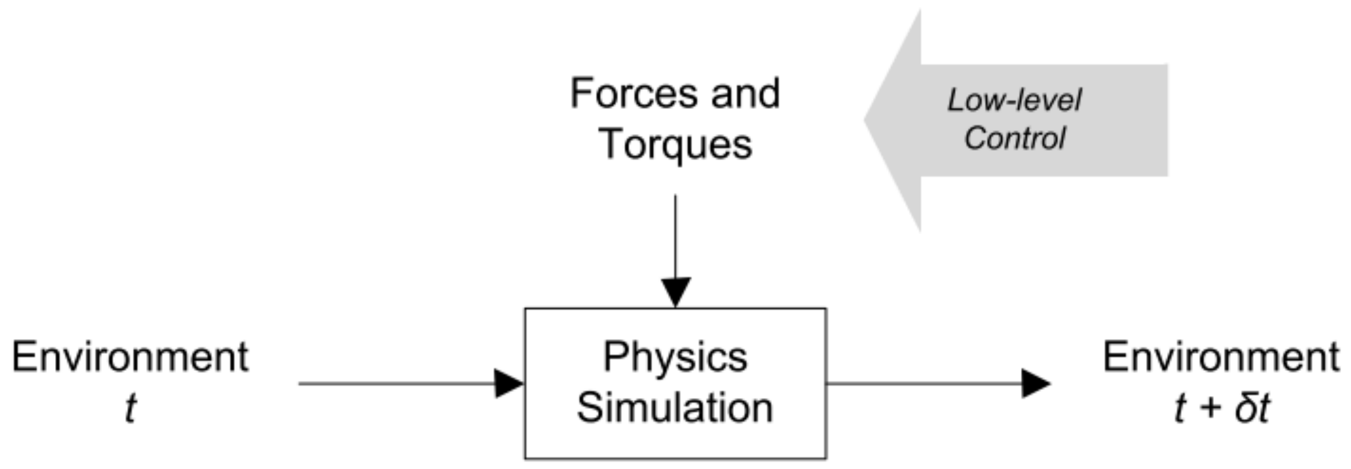


Figure 2: *Animation using physics simulation.*

Physics Simulation

- Physics simulators consist of the following three steps

1. **Collision detection** investigates if intersections exist between different object geometries, and computes information on how to prevent further intersection.

Physics Simulation

- Physics simulators consist of the following three steps

1. **Collision detection** investigates if intersections exist between the different object geometries, and computes information on how to prevent further intersection.

2. **Forward dynamics** computes the linear and angular acceleration of each simulated object, considering external forces and torques, and constraints.

Physics Simulation

- Physics simulators consist of the following three steps

1. **Collision detection** investigates if intersections exist between the different object geometries, and computes information on how to prevent further intersection.

2. **Forward dynamics** computes the linear and angular acceleration of each simulated object, considering external forces and torques, and constraints.

3. **Numerical integration** updates positions, rotations and velocities of objects, based on the accelerations found by forward dynamics.

Physics Simulation

- Physics simulators consist of the following three steps

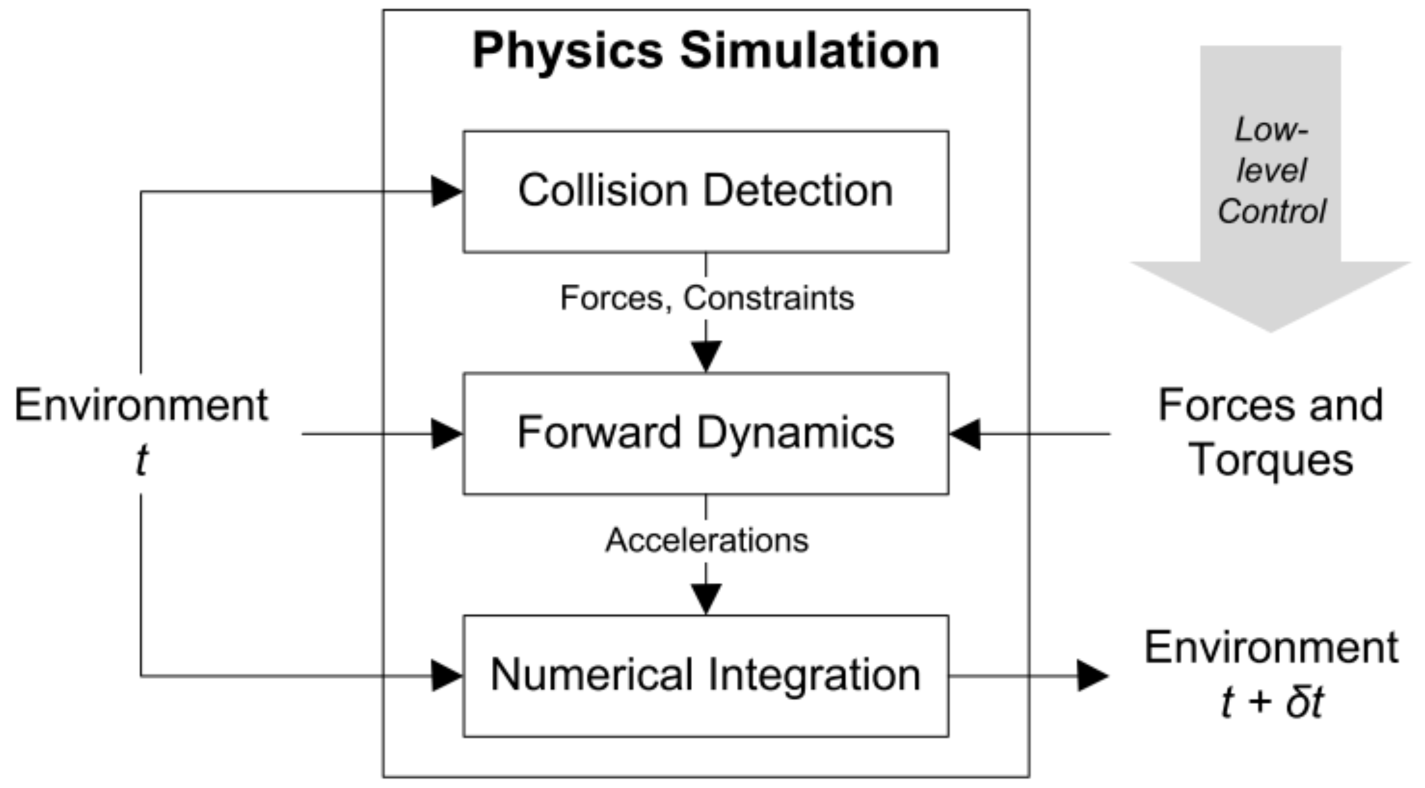


Figure 3: *Animation using physics simulation (detailed).*

Time series and time step

- As we know, computers deal with discrete numbers and states; if the simulation runs sufficiently fast then we see these states as continuous movement.

Time series and time step

- As we know, computers deal with discrete numbers and states; if the simulation runs sufficiently fast then we see these states as continuous movement.
- For each frame of the simulation, we need to calculate the state of each object (i.e. the velocity and position), based on the state of the object in the previous frame.

Time series and time step

- As we know, computers deal with discrete numbers and states; if the simulation runs sufficiently fast then we see these states as continuous movement.
- For each frame of the simulation, we need to calculate the state of each object (i.e. the velocity and position), based on the state of the object in the previous frame.
- The record of a particular state of an object (for example the x coordinate of the object's position) is known as a **time series**.

Time series and time step

- As we know, computers deal with discrete numbers and states; if the simulation runs sufficiently fast then we see these states as continuous movement.
- For each frame of the simulation, we need to calculate the state of each object (i.e. the velocity and position), based on the state of the object in the previous frame.
- The record of a particular state of an object (for example the x coordinate of the object's position) is known as a **time series**.
- The amount of simulated time that has elapsed between each step of the simulation is known as the **time step(δt)** . Note the use of the term simulated time – the time step does not have to be the same as the amount of time it takes to calculate each frame of the simulation. Rather, it is common (and recommended) practice that the time step be a fixed value (e.g., 8ms or 120fps)

Particle Dynamics

Particles and Particle systems

- Particles are objects that have mass, position, and velocity, and respond to forces, but that have no spatial extent.

Particles and Particle systems

- Particles are objects that have mass, position, and velocity, and respond to forces, but that have no spatial extent.
- Because they are simple, particles are by far the easiest objects to simulate.

Particles and Particle systems

- Particles are objects that have mass, position, and velocity, and respond to forces, but that have no spatial extent.
- Because they are simple, particles are by far the easiest objects to simulate.
- Despite their simplicity, particles can be made to exhibit a wide range of interesting behavior. For example, a wide variety of nonrigid structures can be built by connecting particles with simple damped springs.

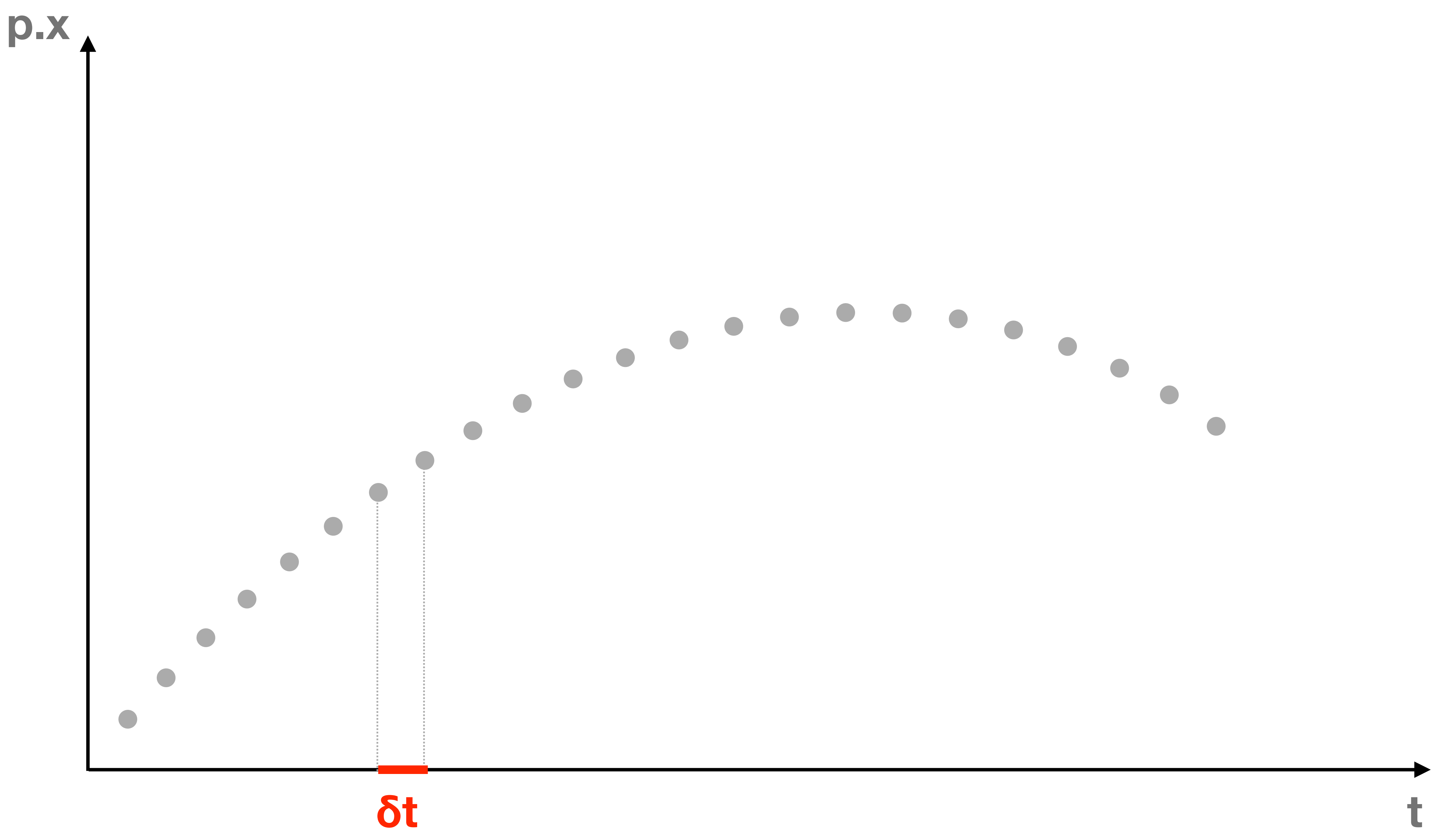
Properties of a particle necessary for physics simulation

- **State** of a particle consists of its **position *p*** and **linear velocity *v***.
- For physics simulation the **mass *m*** of a particle should also be considered.
- Another needed quantity is the **total net force f** acting on the particle.

Physic simulation

- Given the state(position and velocity) and mass of the particle and the net force *f* acting on the particle at time *t*, how do we calculate the state of the particle at time $t+\delta t$?

Time series representation of p.x



Forward Dynamics

1. **Forward dynamics**: Calculate the linear acceleration *a* of the particle.

With Newton's second law of motion, *a* can be easily calculated as below:

$$a = \frac{f}{m}$$

Numerical Integration

As a result of forward dynamics, we now have the acceleration of the particle, so now the problem is how do we calculate the position and velocity of the particle at next time step?

Numerical Integration

As a result of forward dynamics, we now have the acceleration of the particle, so now the problem is how do we calculate the position and velocity of the particle at next time step?

Velocity v is the rate of change of position x over time

$$v = \frac{dx}{dt}$$

Numerical Integration

As a result of forward dynamics, we now have the acceleration of the particle, so now the problem is how do we calculate the position and velocity of the particle at next time step?

Velocity v is the rate of change of position x over time

$$v = \frac{dx}{dt}$$

Acceleration a is the rate of change of velocity v over time

$$a = \frac{dv}{dt}$$

Numerical Integration

As a result of forward dynamics, we now have the acceleration of the particle, so now the problem is how do we calculate the position and velocity of the particle at next time step?

Velocity v is the rate of change of position x over time

$$v = \frac{dx}{dt}$$

Acceleration a is the rate of change of velocity v over time

$$a = \frac{dv}{dt}$$

Conversely, velocity is the integral of acceleration over time, and displacement is the integral of velocity over time

$$\mathrm{v} = \int_t^{t+\delta t} a \ \mathrm{dt} \qquad \mathrm{x} = \int_t^{t+\delta t} v \ \mathrm{dt}$$

Numerical Integration

However we don’t have an explicit parametric formulation for acceleration or velocity over time, we only have discrete values of positions and velocities

Numerical Integration

However we don’t have an explicit parametric formulation for acceleration or velocity over time, we only have discrete values of positions and velocities

Moreover, even we have these parametric formulations, taking integrals or derivatives requires symbolic programming, which is not efficient to use for complex systems and functions.

Numerical Integration

However we don’t have an explicit parametric formulation for acceleration or velocity over time, we only have discrete values of positions and velocities

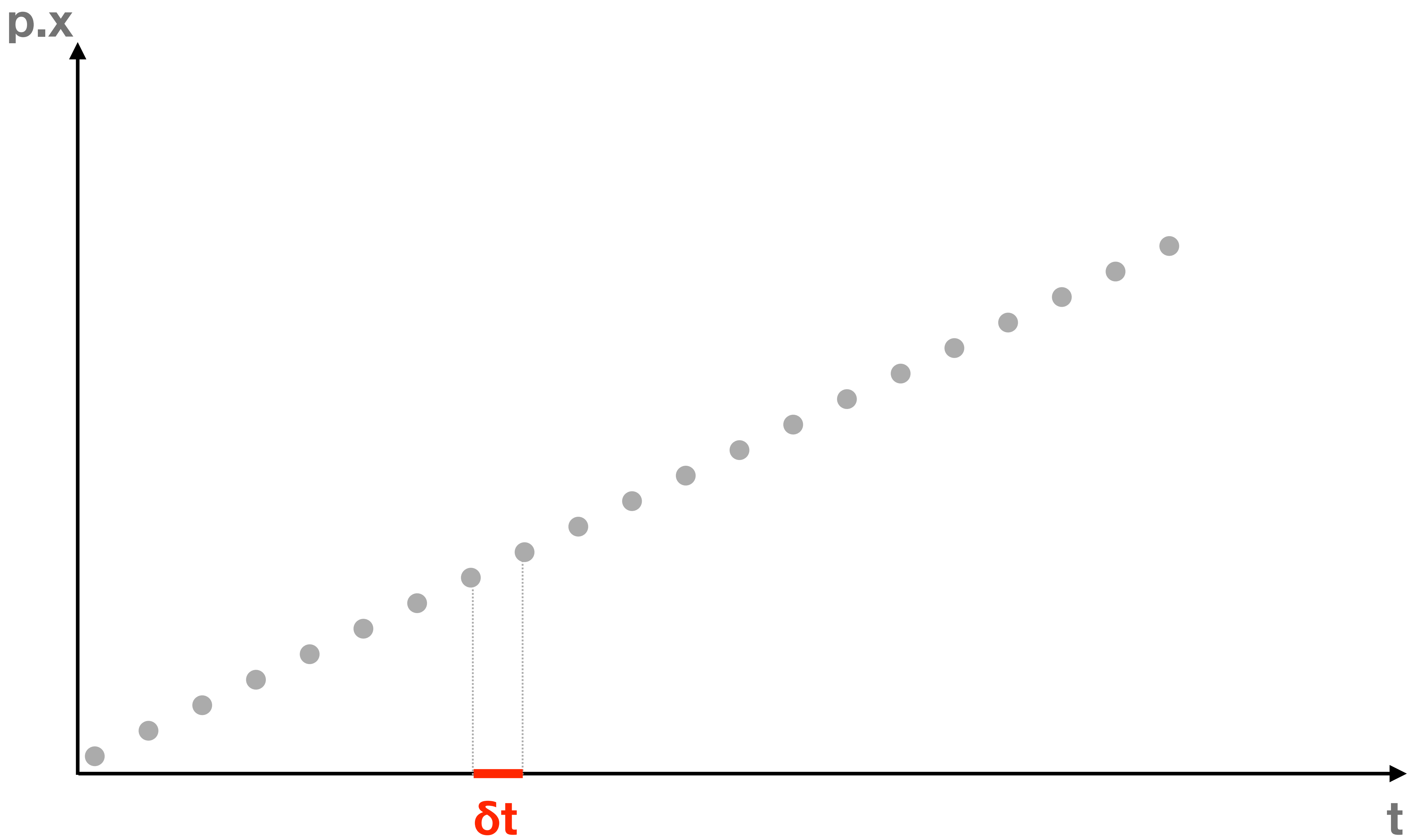
Moreover, even we have these parametric formulations, taking integrals or derivatives requires symbolic programming, which is not efficient to use for complex systems and functions.

Therefore numerical integration methods are used for calculating the next state of an object in the scene.

Explicit Euler Integration

If acceleration was zero, i.o.w. the velocity of the particle was constant, then the time series of the position would be like below and

$$x_{t+\delta t} = x_t + \delta t v_t$$



Explicit Euler Integration

However, most of the time we deal with non-zero accelerations, i.o.w. non-constant velocities

In these cases, again we use the same formulation assuming that the velocity is constant during a small time-step.

$$x_{t+\delta t} = x_t + \delta t v_t$$

Explicit Euler Integration

However, most of the time we deal with non-zero accelerations, i.o.w. non-constant velocities

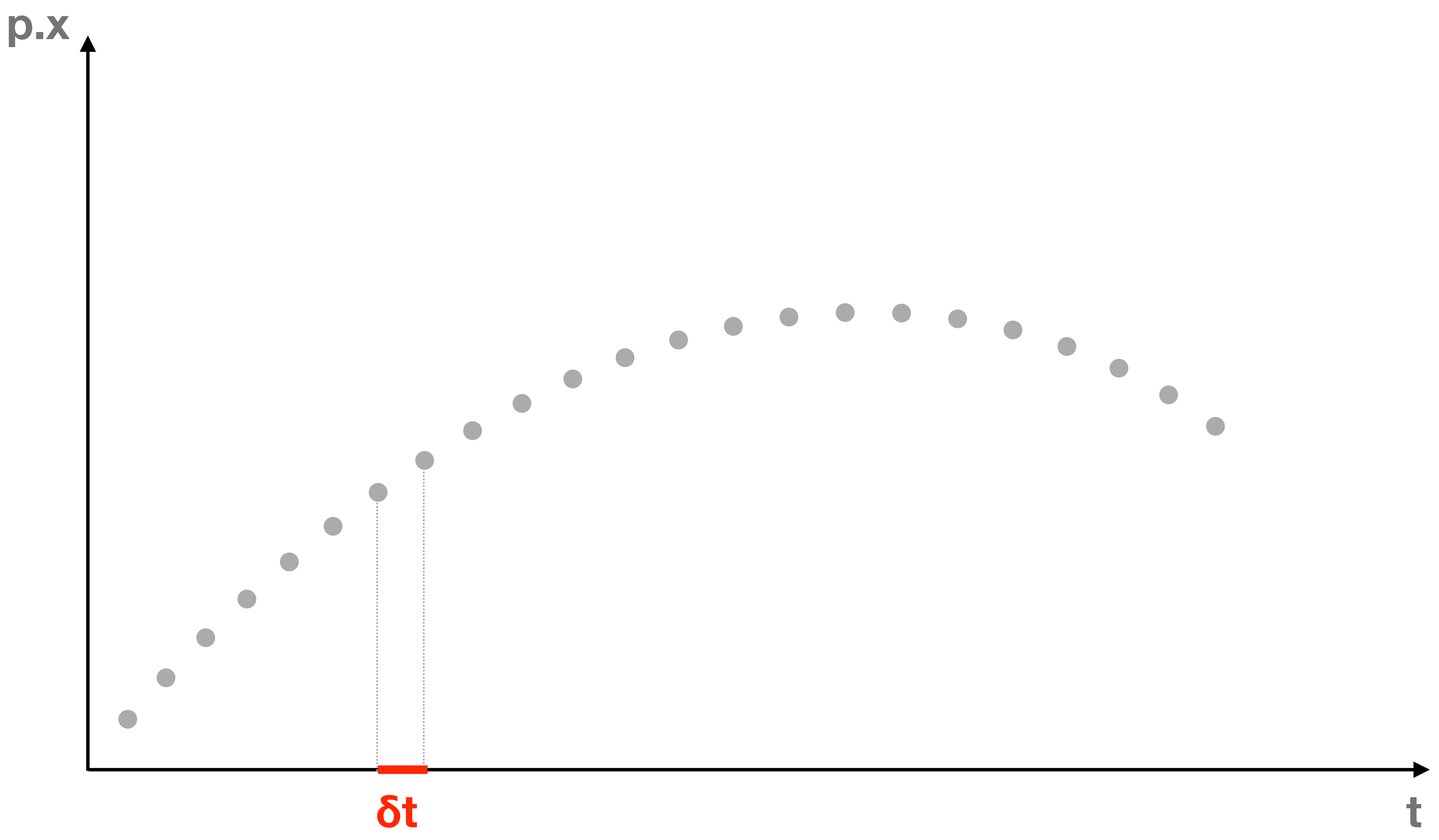
In these cases, again we use the same formulation assuming that the velocity is constant during a small time-step.

$$x_{t+\delta t} = x_t + \delta t v_t$$

And we calculate the velocity at the next time step by using the current velocity and acceleration with the formulation below:

$$v_{t+\delta t} = v_t + \delta t a_t$$

Time series representation of p.x



Accuracy problem of Numerical Integration

Because numerical integration is an attempt to represent a continuous process in a discrete manner, approximations need to be made, which means that the results become inaccurate.

Accuracy problem of Numerical Integration

Because numerical integration is an attempt to represent a continuous process in a discrete manner, approximations need to be made, which means that the results become inaccurate.

As it is an iterative process the inaccuracies are likely to get larger over time (as each result is calculated from the previously inaccurate result, and introduces its own inaccuracy). In the example given, when the velocity of the body is not constant, the assumption that the velocity remains constant throughout a specific time-step is no longer 100% accurate.

Accuracy problem of Numerical Integration

Because numerical integration is an attempt to represent a continuous process in a discrete manner, approximations need to be made, which means that the results become inaccurate.

As it is an iterative process the inaccuracies are likely to get larger over time (as each result is calculated from the previously inaccurate result, and introduces its own inaccuracy). In the example given, when the velocity of the body is not constant, the assumption that the velocity remains constant throughout a specific time-step is no longer 100% accurate.

Of course, **increasing the number of samples (i.e. reducing the time step)** should improve the accuracy. This is why a physics engine typically runs at a much faster frame-rate than the rest of the game.

How to calculate net force acting on a particle?

Types of forces

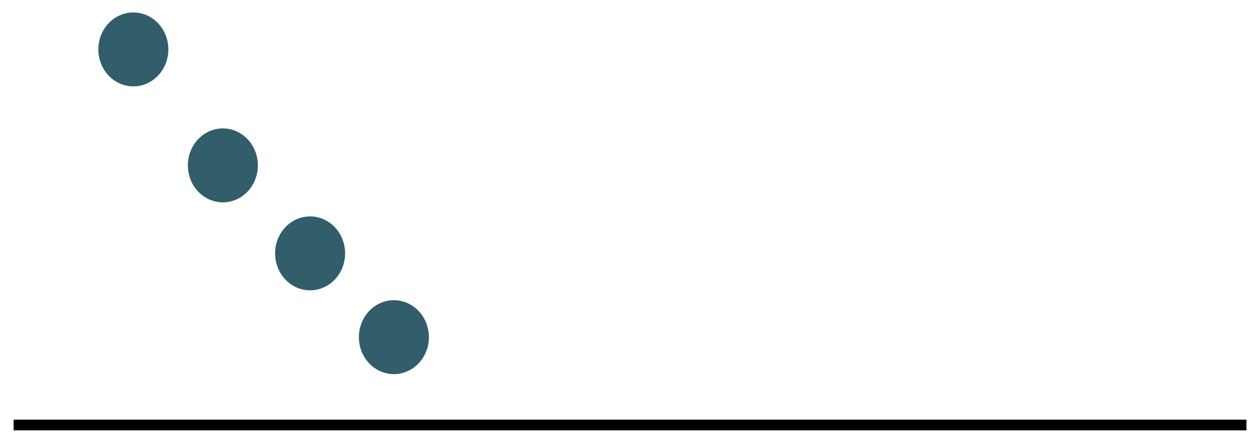
Unary forces: Acting independently on each particle (gravity, drag)

n-ary forces: such as springs, applying forces to a fixed set of particles

Forces of spatial interaction: such as attraction and repulsion, that may act on any or all pairs of particles depending on their positions

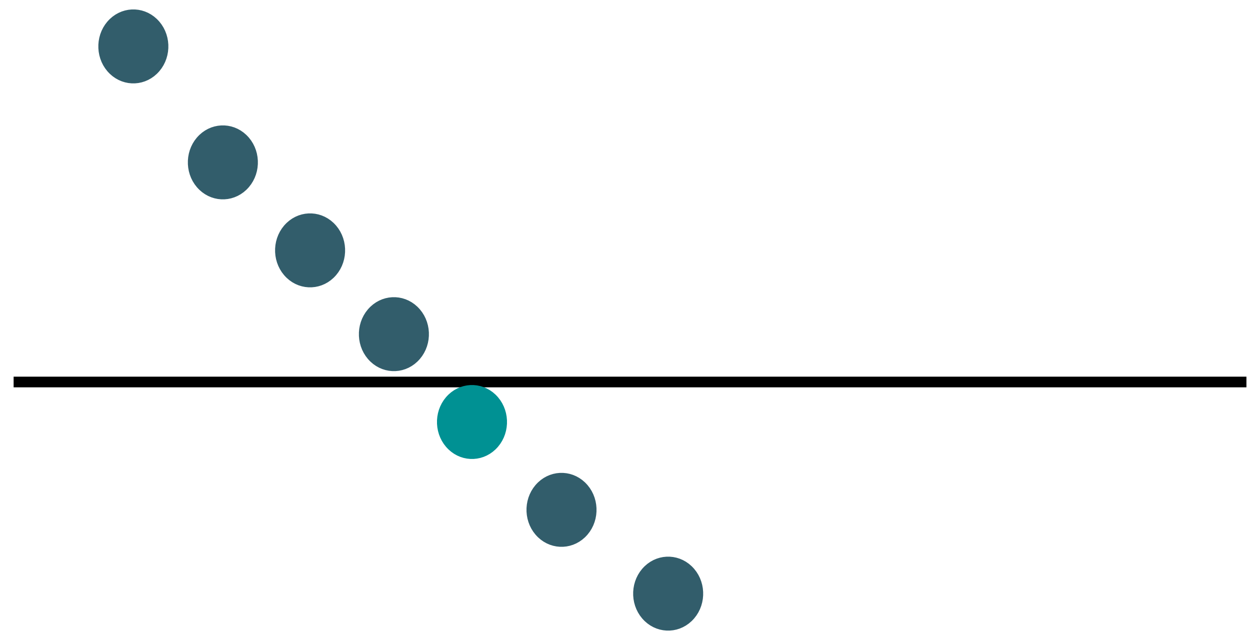
Collision detection of a particle with a planar surface

- Let's first think of a simple scenario, where a particle is falling towards the ground.



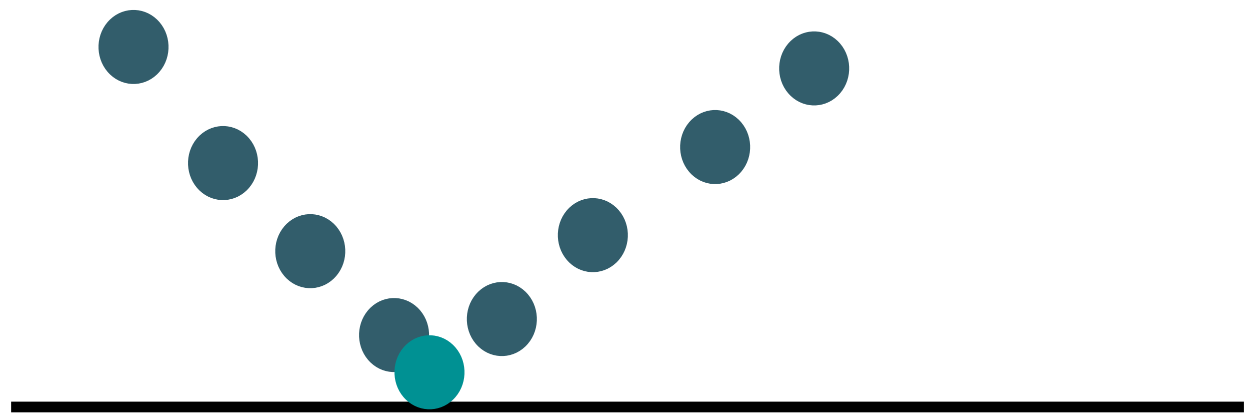
Collision detection of a particle with a planar surface

- If our physics simulator doesn't have a collision detection system, then the particle just passes through the ground:



Collision detection of a particle with a planar surface

- However, we want it to collide with the ground and bounce back with an amount dependent on its bounciness

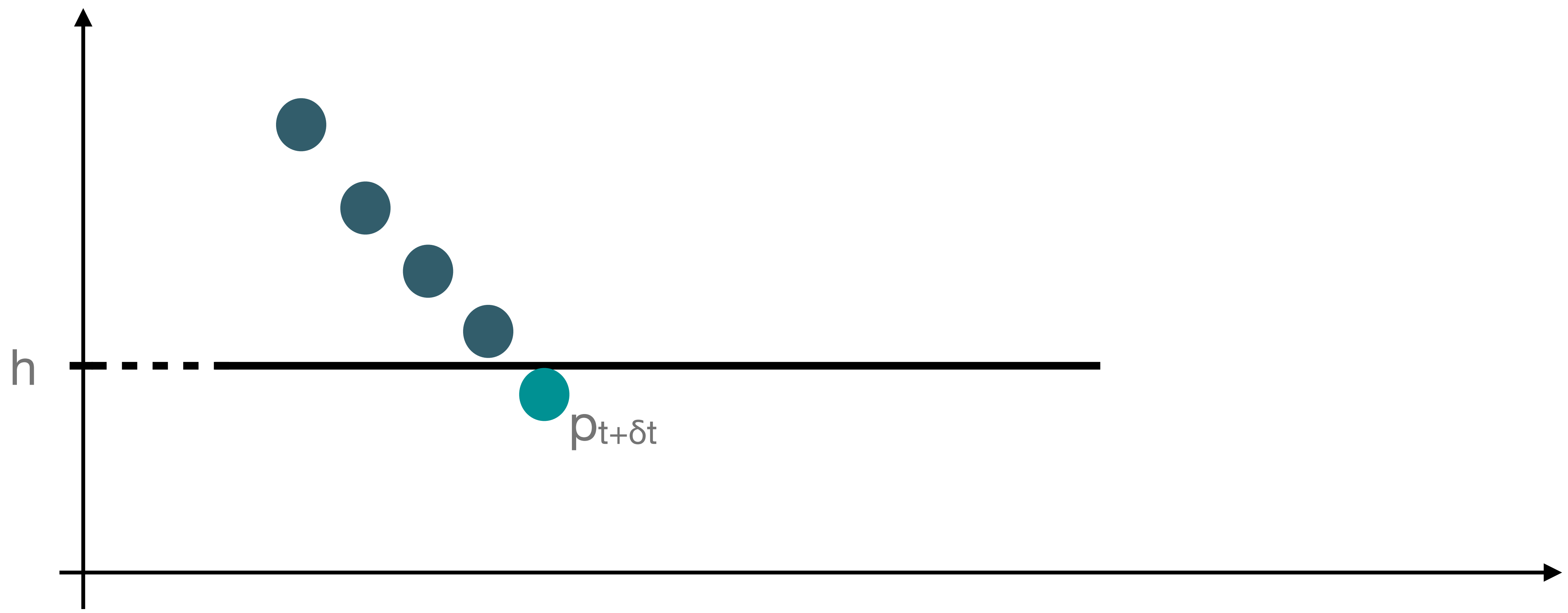


Collision detection of a particle with a planar surface

- For realistic collision response, we should detect the upcoming collision before penetration happens

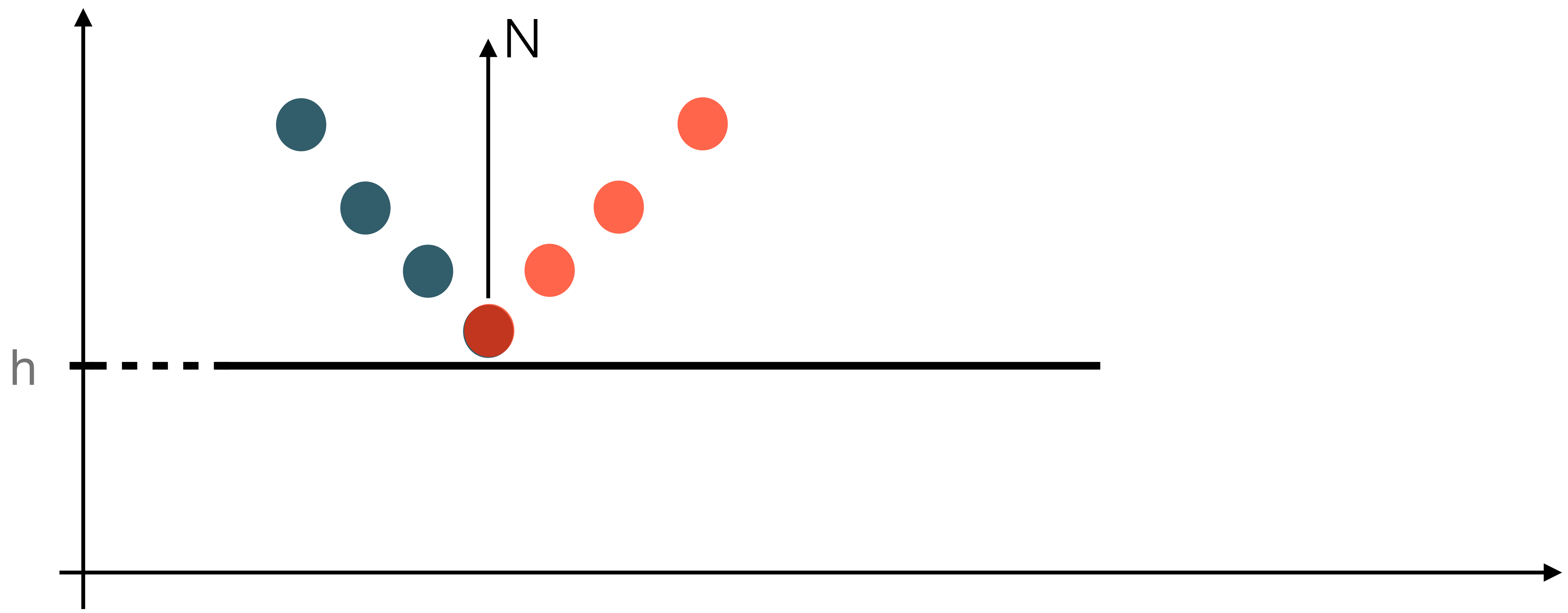
1) Calculate the position of the particle at next time-step; $p_{t+\delta t}$

2) If $p_{t+\delta t}.y < h$, then collision will happen at the next time step.



Collision detection of a particle with a planar surface

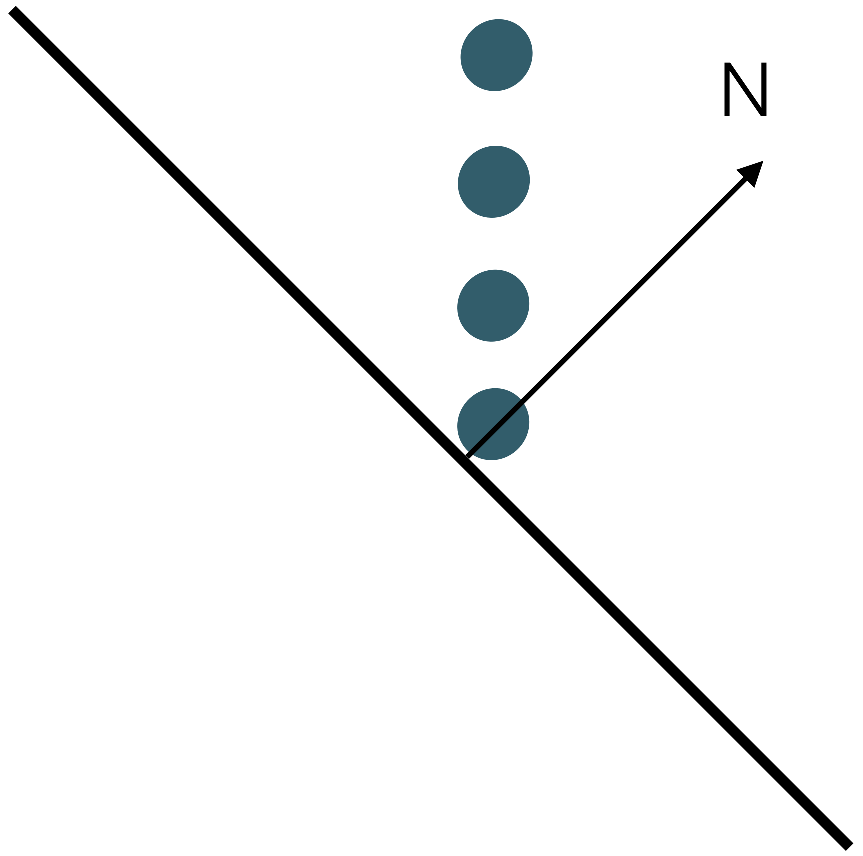
- If the surface of the ground is frictionless and the particle is perfectly bouncy then the particle would behave like the reflection of the light from a mirror.



Collision detection of a particle with a planar surface

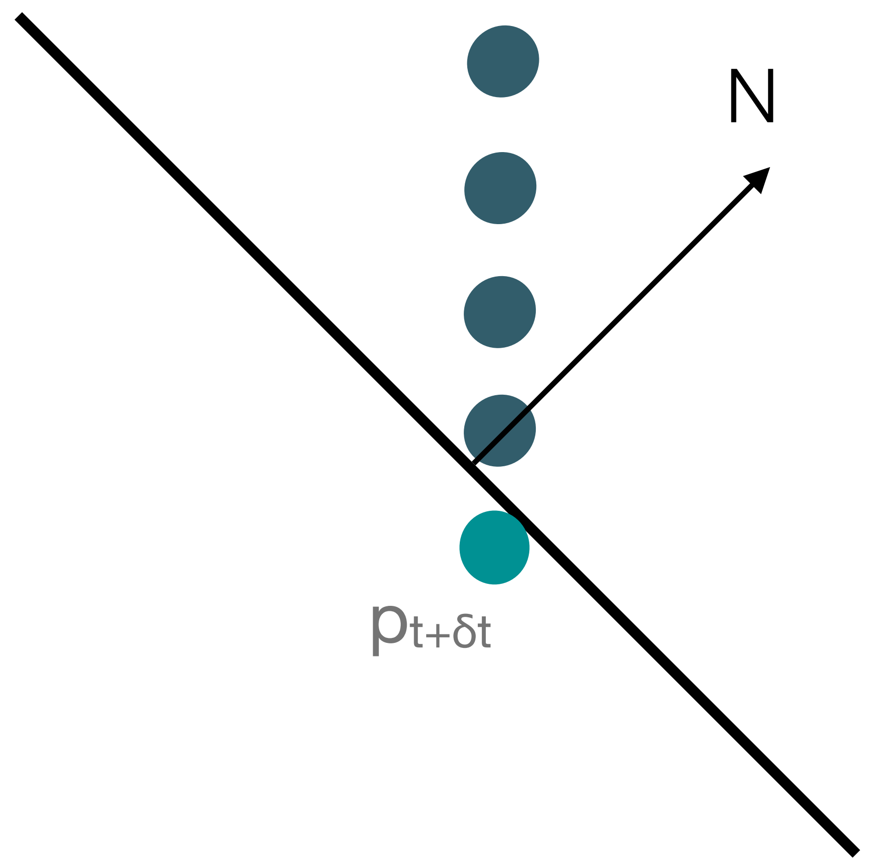
- The collision detection and collision response calculations can be generalized to all kind of planes by using the normal vector N of the plane.

Collision Detection



Collision detection of a particle with a planar surface

- The collision detection and collision response calculations can be generalized to all kind of planes by using the normal vector N of the plane.

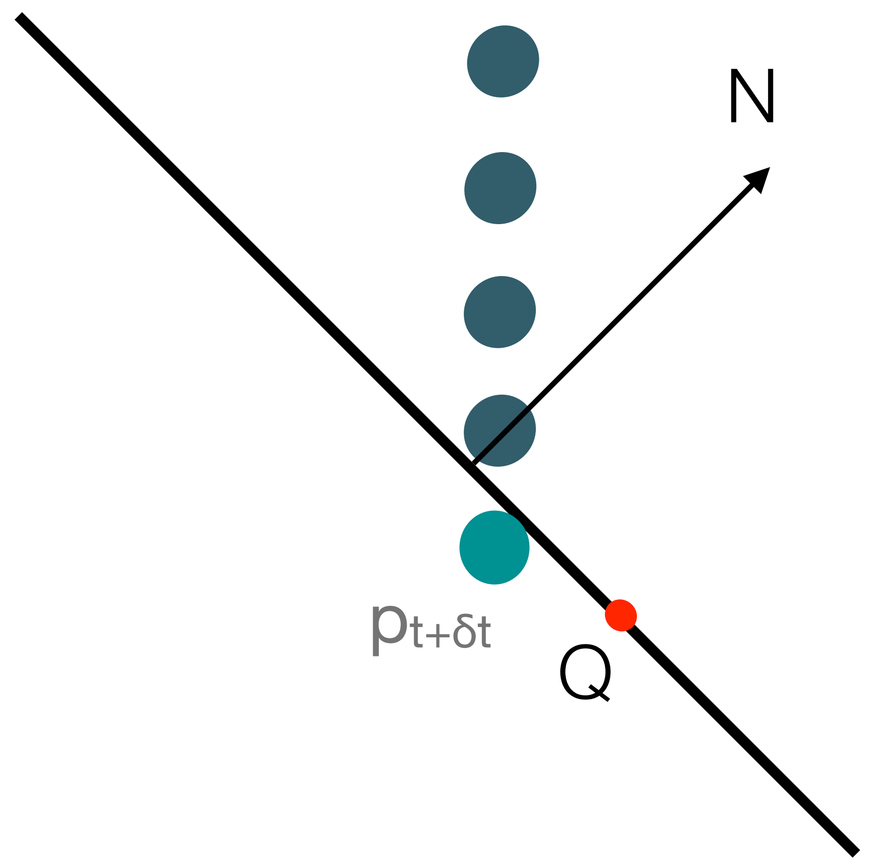


Collision Detection

1) Calculate the position of the particle at next time-step; $p_{t+\delta t}$

Collision detection of a particle with a planar surface

- The collision detection and collision response calculations can be generalized to all kind of planes by using the normal vector N of the plane.



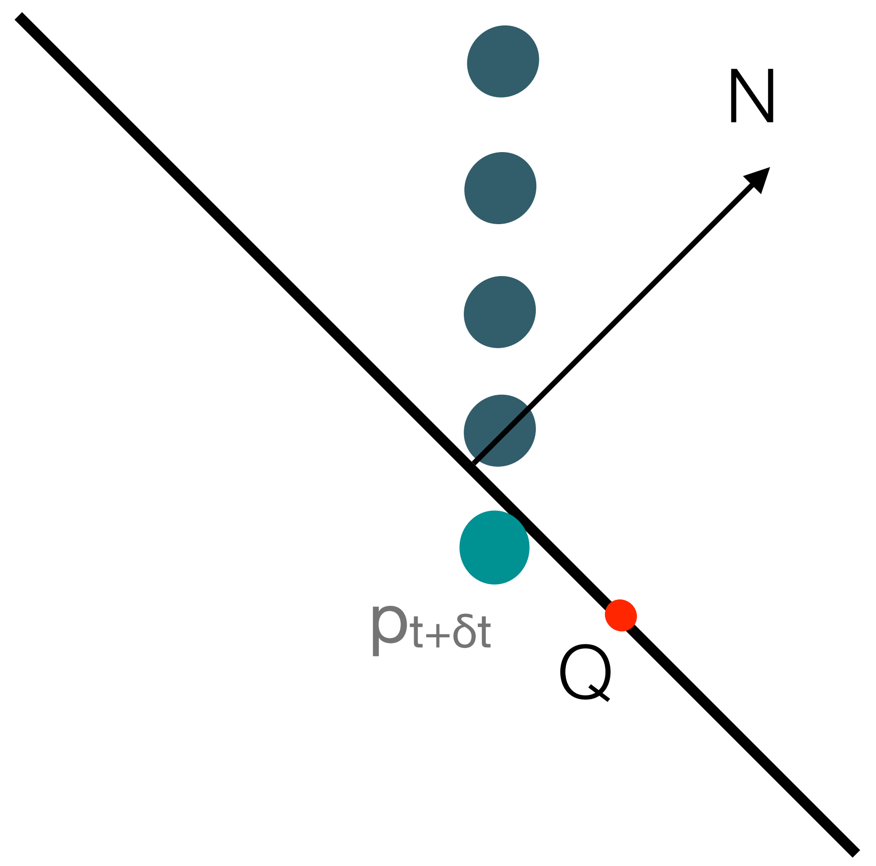
Collision Detection

1) Calculate the position of the particle at next time-step; $p_{t+\delta t}$

2) Pick a point Q on the plane

Collision detection of a particle with a planar surface

- The collision detection and collision response calculations can be generalized to all kind of planes by using the normal vector N of the plane.



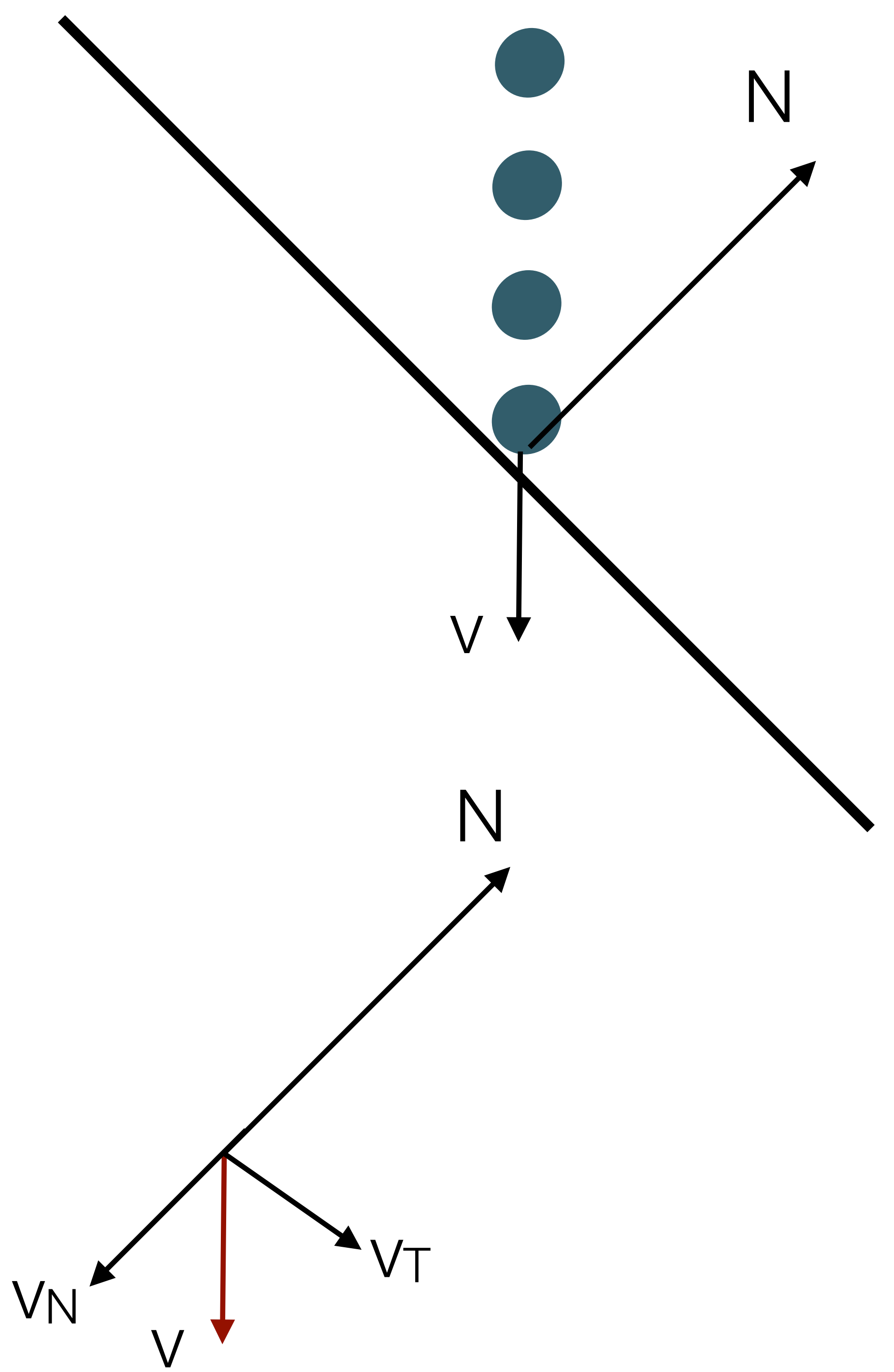
Collision Detection

1) Calculate the position of the particle at next time-step; $p_{t+\delta t}$

2) Pick a point Q on the plane

3) if $(p_{t+\delta t}-Q) \cdot N < 0$, then particle and plane will collide at the next time step

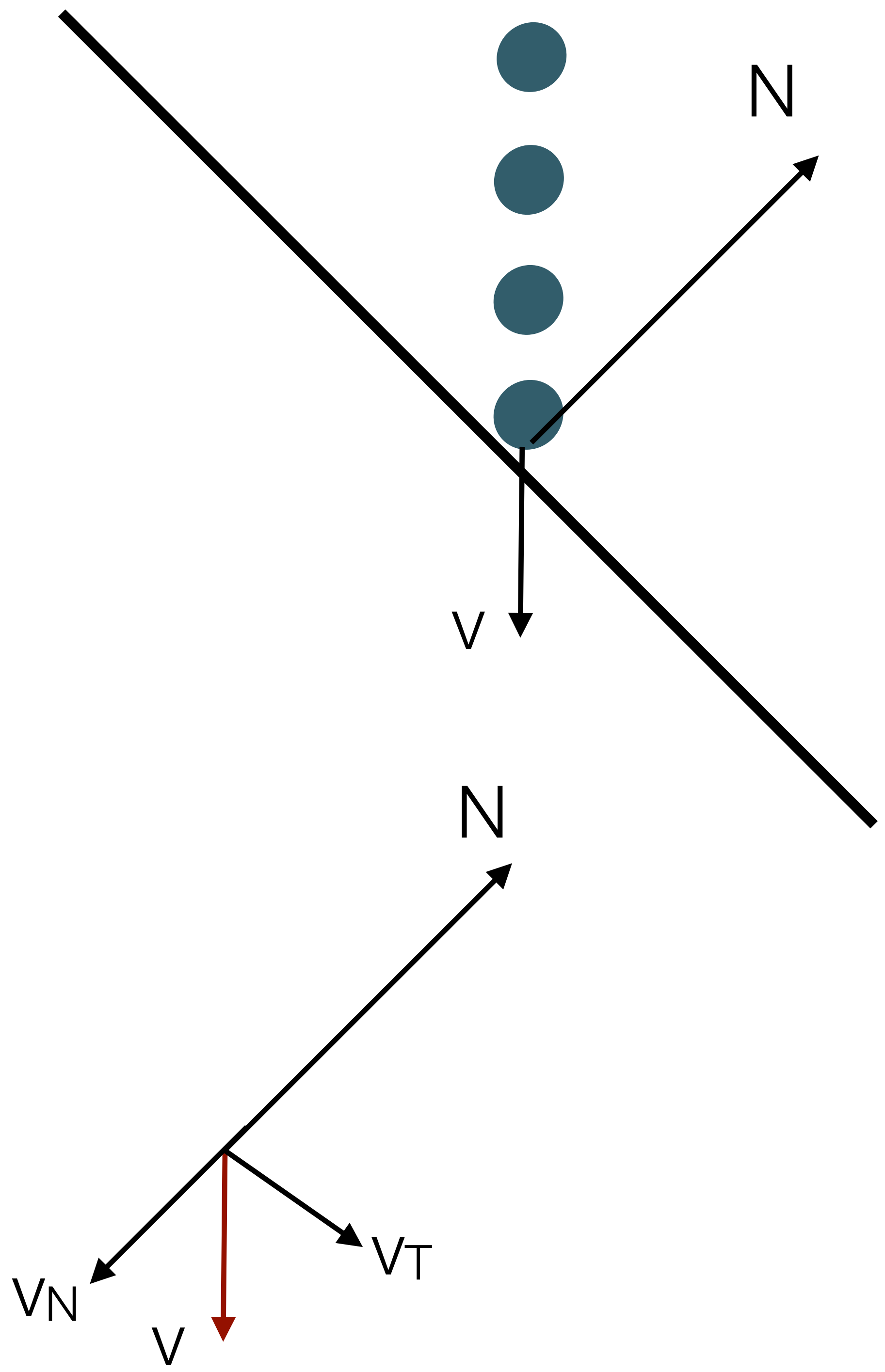
Collision detection of a particle with a planar surface



Collision Response

The velocity vector v is the sum of two vectors, the tangential component v_T and the normal component v_N, where $v_T \perp N$ and $v_N \parallel N$

Collision detection of a particle with a planar surface



Collision Response

The velocity vector v is the sum of two vectors, the tangential component v_T and the normal component v_N, where $v_T \perp N$ and $v_N \parallel N$

Then the velocity of the particle after collision would be

$$V' = (1 - k_f)V_T - k_b V_N$$

Here k_f is a scalar value between 0 and 1 denoting the friction of the surface.

And k_b is also a scalar value between 0 and 1 denoting the bounciness of the particle.

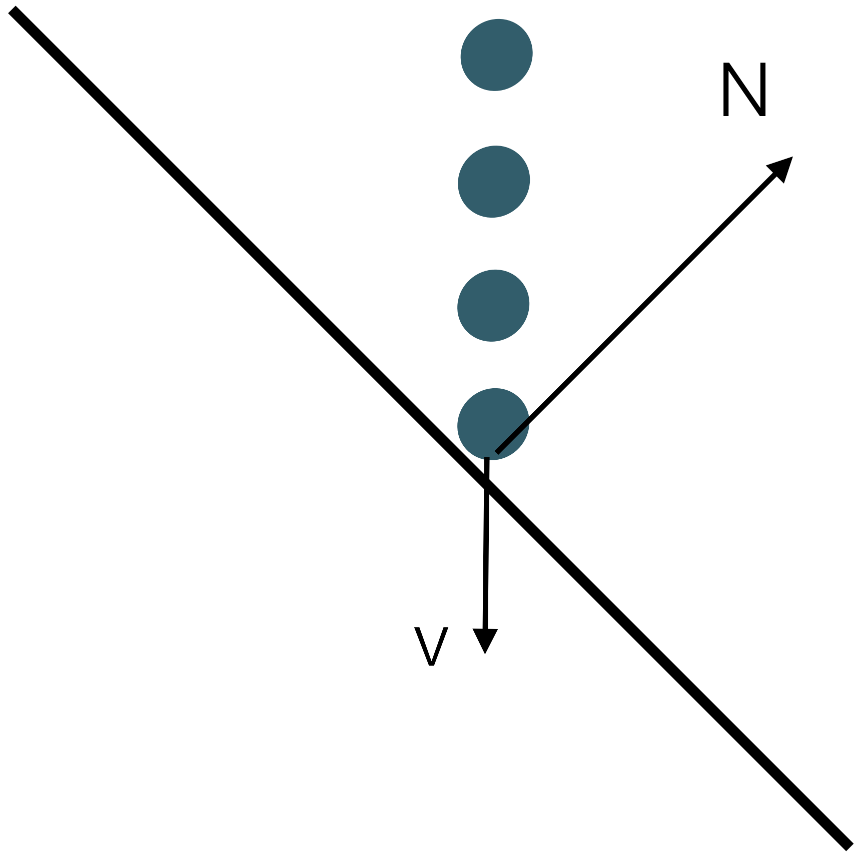
Friction and bounciness

- Physics simulations commonly use the **Coulomb friction model**, which requires a coefficient of friction between the materials of two interacting objects to calculate the friction force.
- Similarly, a **coefficient of restitution(bounciness coefficient)** is required to calculate the collision response force.

Friction and bounciness

$$V' = (1 - k_f)V_T - k_b V_N$$

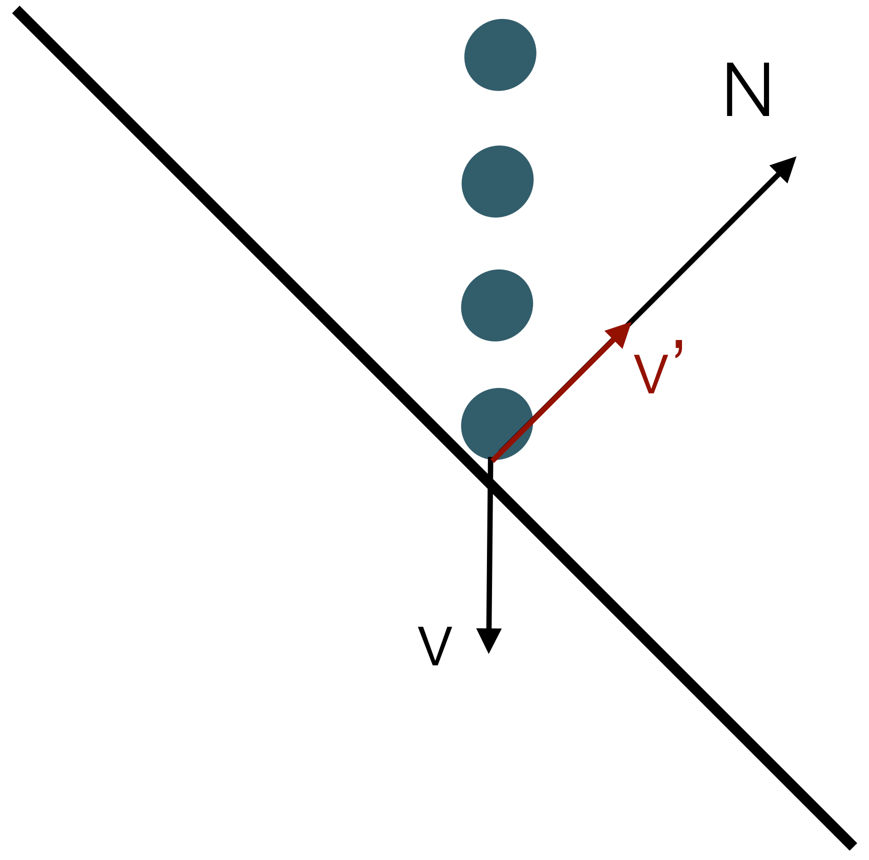
$k_f = 1$



Friction and bounciness

$$V' = (1 - k_f)V_T - k_b V_N$$

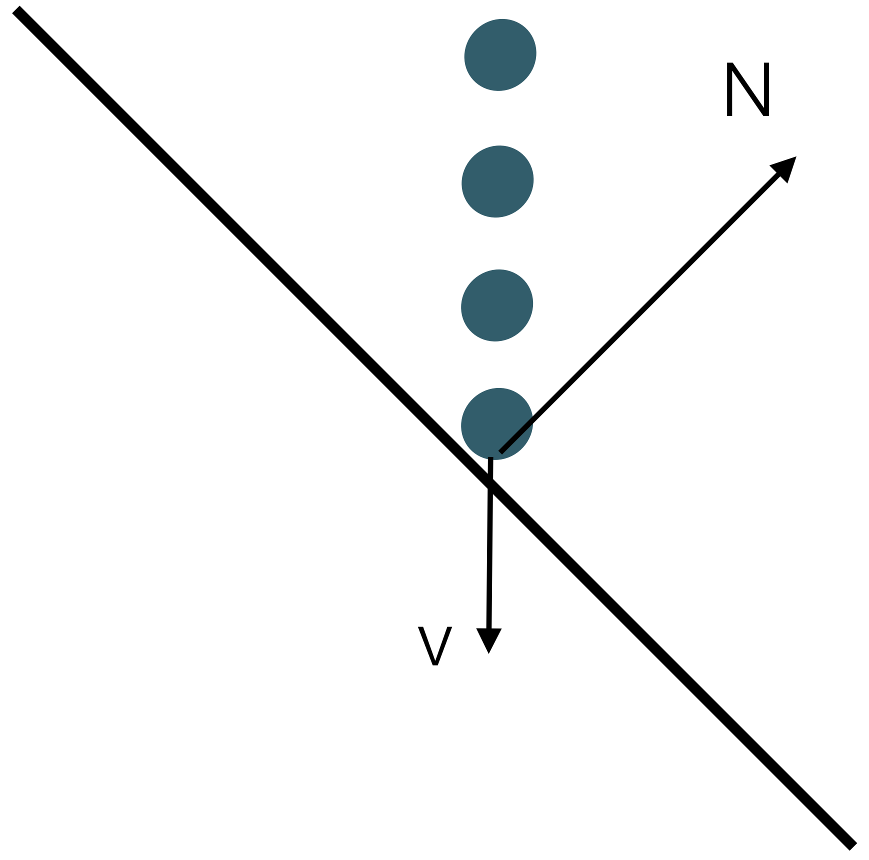
$k_f = 1$ means the surface is perfectly grippy and V' will be parallel to N.



Friction and bounciness

$$V' = (1 - k_f)V_T - k_b V_N$$

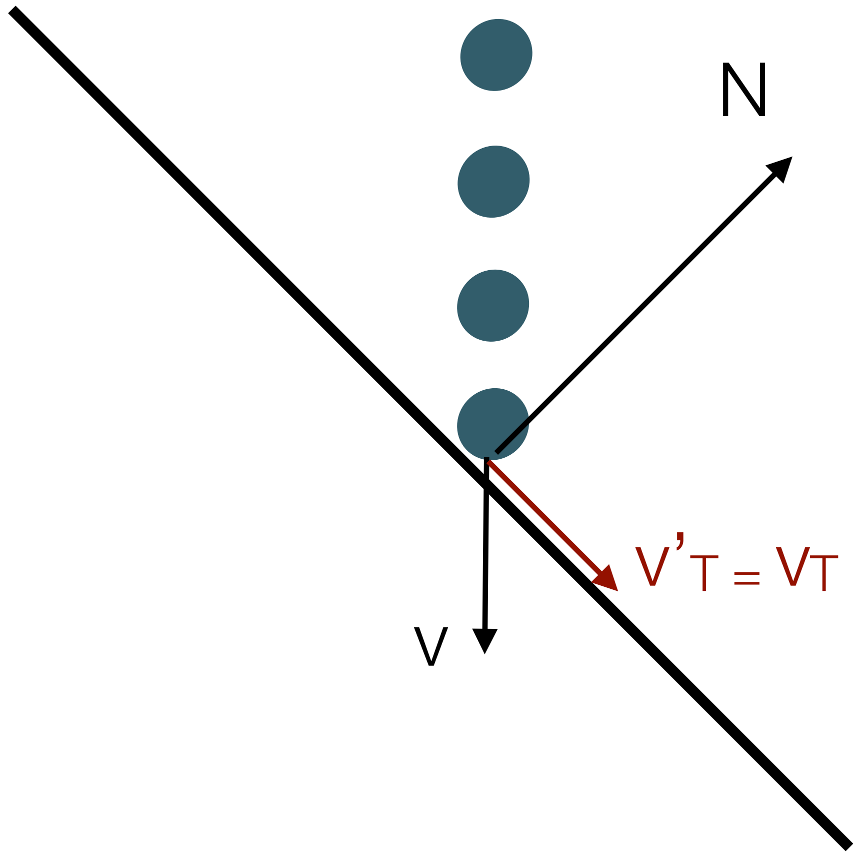
$k_f = 0$



Friction and bounciness

$$V' = (1 - k_f)V_T - k_b V_N$$

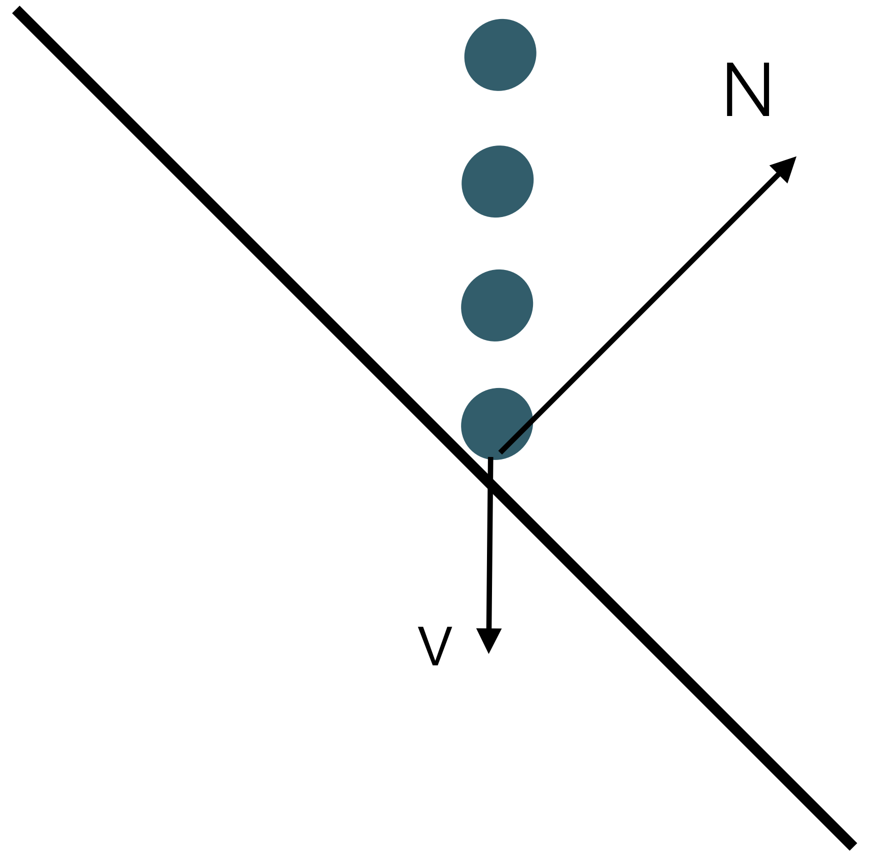
$k_f = 0$ means the surface is perfectly slippery and the particle won't loose any tangential velocity



Friction and bounciness

$$V' = (1 - k_f)V_T - k_b V_N$$

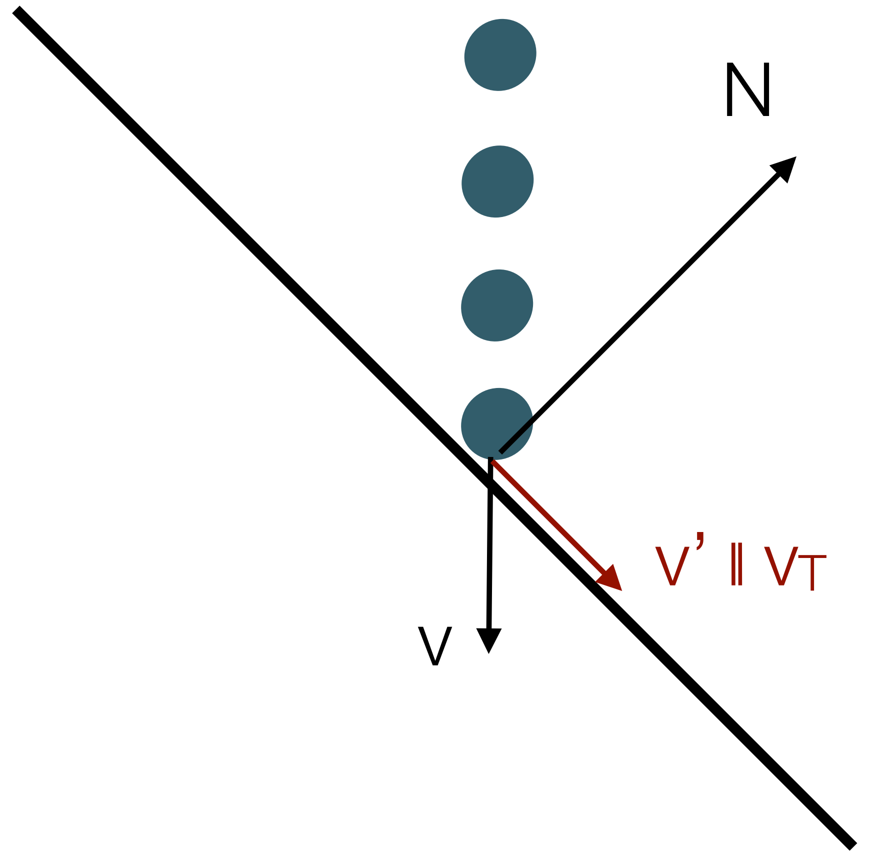
k_b = 0



Friction and bounciness

$$V' = (1 - k_f)V_T - k_b V_N$$

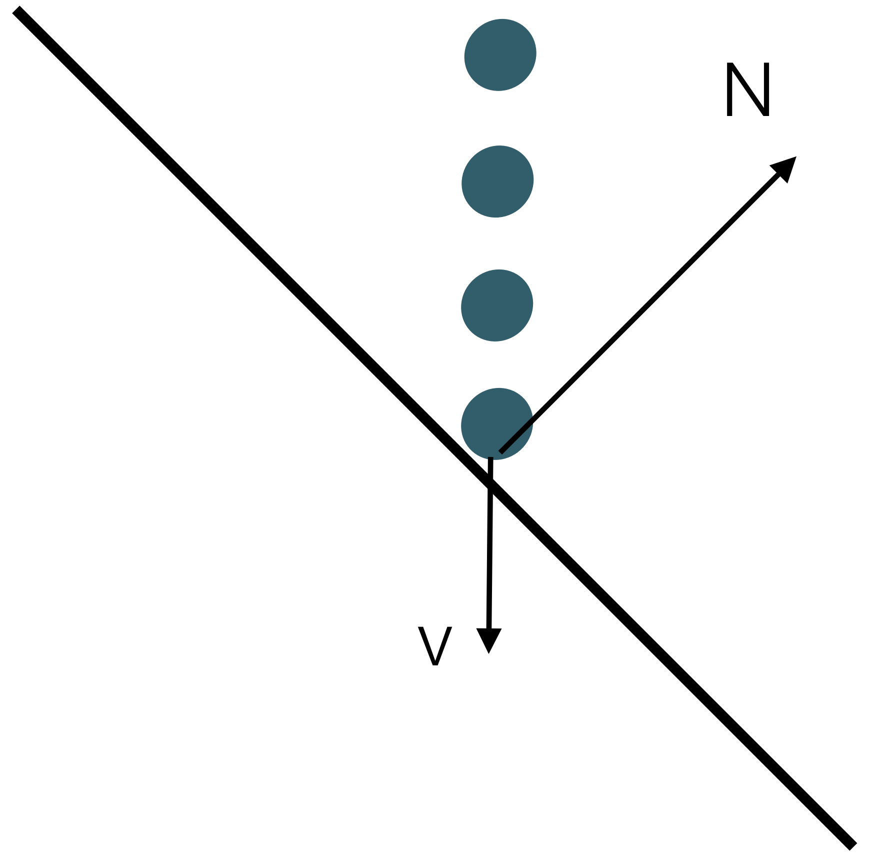
k_b = 0 means the particle has no elasticity, therefore it won't bounce even a little, and V' will be parallel to v_T



Friction and bounciness

$$V' = (1 - k_f)V_T - k_b V_N$$

k_b = 1



Friction and bounciness

$$V' = (1 - k_f)V_T - k_b V_N$$

k_b = 1 means the particle is perfectly elastic, therefore its normal component will be equal to N